SZAKDOLGOZAT

Cseh Miklós Zsolt

Debrecen

2009

Debreceni Egyetem

Informatikai kar

A beszéd fonetikai komponenseinek számítógépes vizsgálata

Témavezető:
Dr Hunyadi László
Egyetemi tanár

Készítette:
Cseh Miklós Zsolt
PTMLK

Debrecen
2009

1 Tartalomjegyzék

2 Bevezetés

A fonetika a hang alaki tulajdonságaival foglalkozik, témája a beszédhang a maga fizikai valóságában. Szorosan kötődik egyéb tudományágakhoz, mint például a fizikához. Az akusztikai elemzésekhez kézenfekvőnek tűnik a számítástechnikát segítségül hívni. Egy olyan programot szándékoztam létrehozni, amely a nyelvészet segítségére lehet ezen a terülten, könnyedén továbbfejleszthető, és több operációs rendszer alatt is futtatható. Mivel a JAVA a legelterjedtebb multi platformos nyelv, ezért az alkalmazást ezen a nyelven valósítottam meg.
Külön kitérek a fonetika egyik fontos területére, a beszéd szegmentációra. Ez több helyen is felhasználható, például a folyamatos beszédfelismerés egyik bemenete lehet.
Az elmúlt években többször is tanúi lehettünk a beszédfelismerés előtérbe kerülésének, de igazi áttörés mind a mai napig nem következett be. Talán ez is mutatja, hogy egyre inkább szükségessé válnak az alternatív beviteli eszközök. A beszédfelismeréssel foglalkozó kutatóknak több problémával kell szembenézniük, mint amire számítottak. Jelenleg az újgenerációs mobiltelefonok terjedésének vagyunk tanúi, melyek maguk után vonják a beszéd alapú alkalmazásokat.
A beszédfelismerés témakörével már 1950 óta foglalkoznak. A kezdeti időkben úgy képzelték el, hogy a megoldáshoz elegendő egy egyszerű Fourier-elemzés. A kutatások előrehaladtával azonban mind több és több akadályt kellett leküzdeniük a tudósoknak. Elméletek tucatjai születtek. Egyesek szerint, ha közelebb szeretnének kerülni a megoldásokhoz, a felismerés során az emberben lejátszódó folyamatokat kell alapul venni. Ezen folyamatok is vita tárgyát képezik a szakemberek körében.
A dolgozat részletezésénél figyelembe vettem azt, hogy nemcsak informatikusok, és matematikusok, hanem nyelvészek is olvashatják. Úgy próbáltam strukturálni szakdolgozatomat, hogy az első fele könnyebben érthető legyen. Igyekeztem a szükséges algoritmusokat képletek, egyenletek nélkül leírni. A matematikai hátteret egy külön fejezet tartalmazza.
Először a beszéd modelljéről: a hangok a hallás jellemzőiről és a beszédkeltésről érzékelésről adok egy rövid áttekintést. Ezt követi a jelfeldolgozás alapjairól szóló fejezet, ahol megismerkedhetünk a mintavételezéssel, a fonetikában használt néhány diagrammal, a Fourier és koszinusz transzformációkkal és e transzformációk során használt ablakokkal. Ezek

a fejezetek megalapozzák a szegmentációról szóló részeket, ahol először a nyelvészeti majd a számítógépes támogatásról lesz szó.

Miután a szükséges elméleti hátteret kifejtettem röviden foglalkozom az elkészült alkalmazással is. Néhány lényegesebb kódot a függelékben mellékelek, de az egész program terjedelmi okokból túl lépi a dolgozat kereteit.

Dolgozatomnak nem az a célja, hogy „nagy” problémákat oldjon meg, vagy teljes körű áttekintést adjon a mai napig elért eredményekről. Csupán ízelítőt szeretnék nyújtani a számítógéppel támogatott fonetika témaköréből. Az itt leírtak sok helyen tovább gondolhatóak, kiegészíthetőek.

3 A beszéd modellje

3.1 Hangok, hallás

A hangnak nincs precíz definíciója. A hang a közege részecskéinek nyugalmi állapotából való kilengés által keletkezik.

3.1.1 Hang jellemzői:

- Intenzitás. Jele az *I*, egysége a watt/m^2. Az intenzitás az egységnyi keresztmetszeten átáramló energia. Az *I* mindig arányos az amplitúdó négyzetével.
- Szinuszos hang jellemzői (a szinuszos hang alapeleme a hangfeldolgozásnak):
  - Periódusidő, reciproka a frekvencia
  - Amplitúdó
  - Fázis
- Összetett hang fajtái:
  - Kvázi periodikus
  - Zajszerű (monoton zaj pl.: fehér zaj)
  - Impulzusszerű zaj

Azt, hogy hallásunk során mit érzékelünk és milyen pontosan, a Weber-Fechner-féle pszichofizikai törvény írja le:

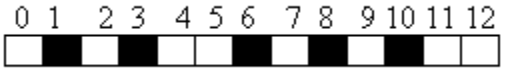
- Érzékelésünk sok esetben (hangmagasság, intenzitás) nem lineáris, hanem logaritmikus.
- A hangmagasság (frekvencia) érzékelésére jellemző, hogy körülbelül 16 Hz-től körülbelül 20 kHz-ig hallunk, de általánosabb a 20Hz – 16KHz
- Érzékeljük az intenzitást.
- Nem érzékeljük a fázist.
- A hangforrás irányát 3° pontossággal tudjuk meghatározni.

3.1.2 Zenei hang:

Két hang *(f_1,f_2)* hangköz = *(f_3,f_4)* hangköz. Akkor érezzük ugyanakkora hangköznek a kettőt, ha $\frac{f_2}{f_1} = \frac{f_4}{f_3}$ Egyenlő hangközű skála egy mértani sorozatú f_i frekvencia sorozat. $\frac{f_2}{f_1} = 2$ ez az

oktáv. Jól hangolt skálában az oktávot 12 egyenlő részre osztjuk és itt egy osztásköz egy félhangköz. A zongorán is így helyezkednek el a hangok. $q = \sqrt[12]{2}$ ez a kvócilis a mértani sorozatban.

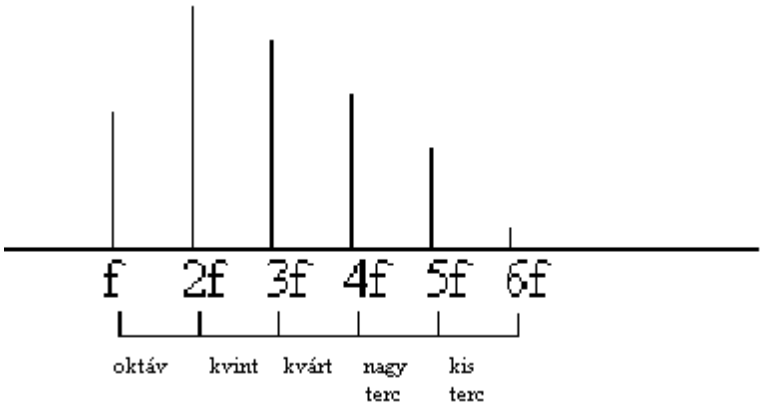
1. Ábra: Egy oktáv



Abszolút hallása van valakinek, ha egy hangról meg tudja mondani milyen hang (dó, ré,…) az ábécés abszolút skálán. Szolmizációs vagy relatív skála esetében a *440* Hz a normál „a”. Ehhez viszonyítjuk a hangokat. Relatív hallással rendelkezik az a személy, aki rögzített hanghoz viszonyítva meg tudja állapítani, hogy milyen a hallott hang. Ez *3%* pontosságú, ha $\left|\frac{f_1}{f_2} - 1\right| < 0{,}03$, akkor összetévesztjük a két hangot. $1{,}03 \approx \sqrt{q}$ ez egy negyed hangnak felel meg. A 16 Hz- 20 000 Hz intervallum egy kicsit több mint 10 oktáv és egy nagy terc. A zongorán 7 oktáv van.

Alaphang *f*, felhang $k \cdot f$ frekvenciájú hangok *k=2,3,*…Vannak olyan hangszerek, amelyeknek nincs alaphangjuk pl.: harang

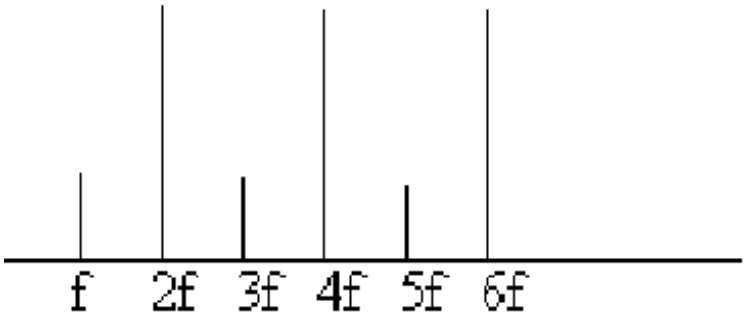
2. Ábra: Tiszta hangközök



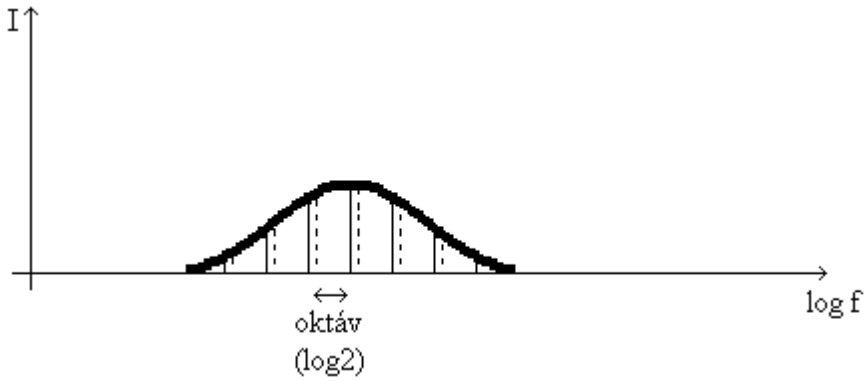
A 2. ábrán a tiszta hangközöket láthatjuk. Pl. egy kvint $\frac{3}{2} \approx 2^{\frac{7}{12}}$. Az oktáv a 8-ból ered. Az alaphangot általában nehéz megszólaltatni. $\frac{20}{16} = \frac{5}{4}$ egy nagy terc (a vájt fülűek ennyivel többet hallanak az alsó határon)

Egy hangmagasság kialakulásához Kb. 3 periódusidő szükséges, de ez frekvenciafüggő. Egyes esetekben előfordulhat, hogy nem az alaphangnak megfelelő hangmagasságot érzékelünk. Például ha minden páratlan felharmonikus alacsony amplitúdójú, akkor egy oktávval magasabbnak érzékeljük a hangot. (Lásd 3. Ábra) A 4. ábrán a Hirosima c. film zenéjében időben mozgatjuk a rácsot egyenletesen, a harang fix, de a hang emelkedik is meg nem is (szirénaféle hang lesz) 1 oktávot megtéve oda jutok, ahonnan indultam.

3. Ábra: Páratlan felharmónikusoknál alacsony amplitúdójú hang.



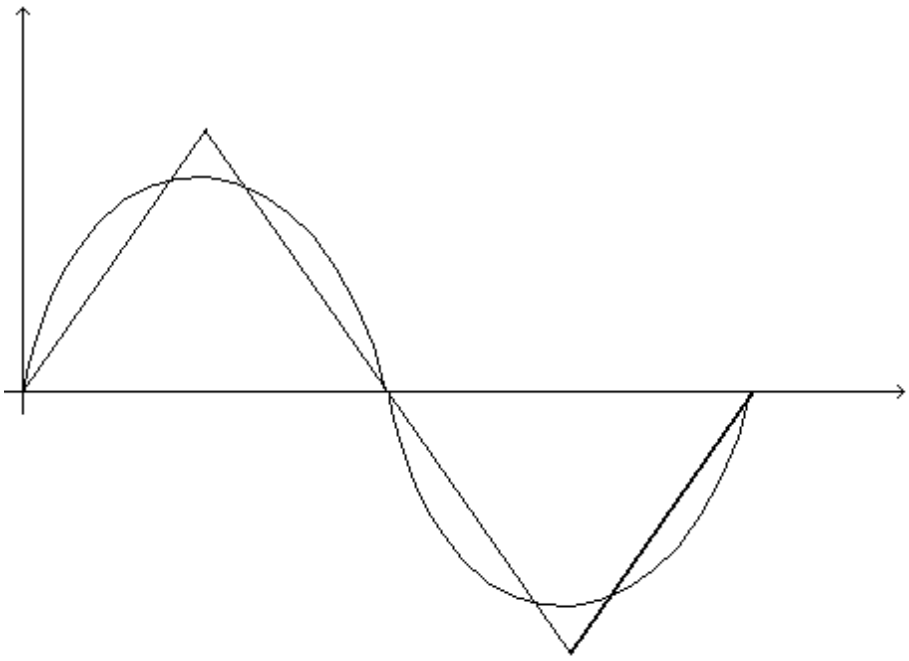
4. Ábra: Hirosima című film zenéje



Hangmagasság fogalom:

- Alaphangos (kiegészítjük felhangokkal a meglévő rendszert).
- Spektrális (intenzitással súlyozott frekvenciaátlag) a Hirosimában a spektrális hangmagasság állandó.
- Fázis érzékelés (fázisviszonyt nem tudjuk érzékelni. 5. Ábrán két füllel megkülönböztethetetlen hang képét láthatjuk. Fázisdiszperzió (fáziskeveredés)).

5. Ábra: Két füllel megkülönböztethetetlen hang képe



- Intenzitás érzékelése

  I az amplitúdó négyzetével arányos (jel négyzetének integrálja) I_0 a hallásküszöb, alsó határ ($I_0 = 10^{-12}$ watt/m^2). A Felső határ a fájdalomküszöb. $10 \lg \frac{I}{I_0}$ SPL hangnyomásszint (egysége a dB)

  A hangerőérzet frekvencia és intenzitásfüggő (hangnyomás szint) h – hangerőérzet $h(f, I)(\neq c \lg I / I_0)$ Az intenzitás mérésére használatos egy szubjektív phon-skála az 1000 Hz-es hang hangnyomásszintjével meghatározott azonos hangerőérzet. Az intenzitást dB-ben mérjük I_0 intenzitású hang amplitúdója az elektronpálya átmérőjének 10-ed része.

Némely esetben fontos szót ejteni az elfedésről (masking), ami azt jelenti, hogy az alhang elfedi a felhangjait. A felhangoknak sokkal erősebben kell szólniuk, hogy az alhangoktól meg lehessen különböztetni.

3.1.3 Mel-skála:

A hangmagasság meghatározásához szintén használnak egy a hangérzetet jobban követő Mel-skálát, amit tapasztalati úton alakítottak ki. Önkéntesek segítségével meghatározták, hogy mi az a frekvencia különbség, amit az emberi fül meg tud különböztetni egymástól, majd az így kapott értékekre illesztettek egy görbét.

Az ember hangérzékelése a magasabb frekvenciákon szélesebb frekvenciaközű hangokat képes csak megkülönböztetni, mint alacsonyabb frekvenciákon. 1000 Hz alatt közelítőleg lineárisan változik, míg e fölött logaritmikusan (tehát 1000 Hz fölött a sávok szélessége a frekvencia függvényében exponenciálisan növekszik). Így a referencia pontot 1000Hz-nek választhatjuk, azaz ez az érték megegyezik a hagyományos és a Mel-skálában.
A hertz mel konverzióra több képlet is ismert. A konkrét képletek a matematikai háttér alatt olvashatóak.

3.2 Beszédkeltés és érzékelés

3.2.1 A beszéd szerkezete

Szöveg - mondat - szintagma - szó - morféma - fonéma (beszédhang) a beszéd agyi szintű atomjai. 250 db fonémát tartanak számon (pl.: 6 db r hangot,…) Association Phonétique Internationale (nemzetközi szabvány szervezet) (API - ilyeneket le lehet tölteni az Internetről) nyelv – fonémakészlet. Ezek az allofonok (minden nyelvnek van saját fonémakészlete)

3.2.2 Hangképzésgerjesztés:

A hangképzés gerjesztés során a tüdőből kiáramló levegő gerjeszti a hangot, ez adja az energiát. Ennek a formái:

- Zöngés: A gégében található hangszálakat rezegtetjük. Ez egy periodikus a legtöbb esetben kváziperiódikus gerjesztés.
- Zörejes: Ezen gerjesztés során turbulens, örvényszerű áramlást hozunk létre, aminek eredményeként a jel aperiodikus lesz. Nyitott hangszálaknál, a szájüreg különböző helyein képzett akadályoknál, szűkületeknél képződik.
- Kevert: Ebben az esetben mind zöngés, mind zörejes gerjesztés jelen van egy időben.
- Lökéshullámszerű: Általában zárfelpattanás modellezésére használják. Ez egy rövid idejű képzés, amit hosszabb néma fázis előz meg.
- Nincs gerjesztés: Fontos szerepet töltenek be, a beszédjel bizonyos részein, például lökéshullámszerű képzést megelőzően.

A hang egy rezonátorrendszerben halad végig, ami szűrőként viselkedik, bizonyos frekvenciákat kiemel, másokat elnyel. Ennek részei a száj-, a garat- és az orrüreg.

„Vizsgáljuk meg a zönge, vagyis a hangszalagrezgéséből keletkező kváziperiodikus rezgés tulajdonságait! A hangszalagok felnyitódása majd összezáródása tulajdonképpen megszaggatja, modulálja a tüdőből kitóduló levegőáramot. A folyamat a következőképpen zajlik le. Az összezárt hangszalagok lezárják a tüdőből kiáramolni kívánó levegő útját. Így a hangszalagok alatti területen a levegőnyomás egyre nagyobb lesz. A nyomás fokozódik, majd egy ponton a hangszalagokat összezáró izmok már nem tudják a zárat tovább fenntartani, így a levegő szétfeszíti a hangszalagokat és kitódul az artikulációs csatornába. A gyors levegőáramlás miatt a hangszalagokat szétfeszítő nyomás lecsökken, a hangszalagokat az izmok újból összezárják. A nyomás ismét fokozódni kezd és a folyamat újból megismétlődik."

„A hangszalagműködést tudatunktól függetlenül a következő tényezők befolyásolják.

- A hangszalag hossza. Ha rövid a hangszalag (gyermekeknél, nőknél), akkor a képzett hang frekvenciája magasabb és fordítva.
- A hangszalag tömege. Vaskosabb hangszalag mozgása lomhább, mint a kisebb tömegűé. A nagyobb tömegű hangszalag rezgésénél a felhangtartalom kisebb.
- A hangszalag rugalmassága. Rugalmas hangszalaggal egészséges, nagy felhangtartalmú zöngét képezhetünk. Ha a hangszalagok rugalmatlanok, akkor a nyitódás-záródás folyamatból kimaradnak a felharmonikust termelő mozzanatok, így a hang színezete tompább lesz.

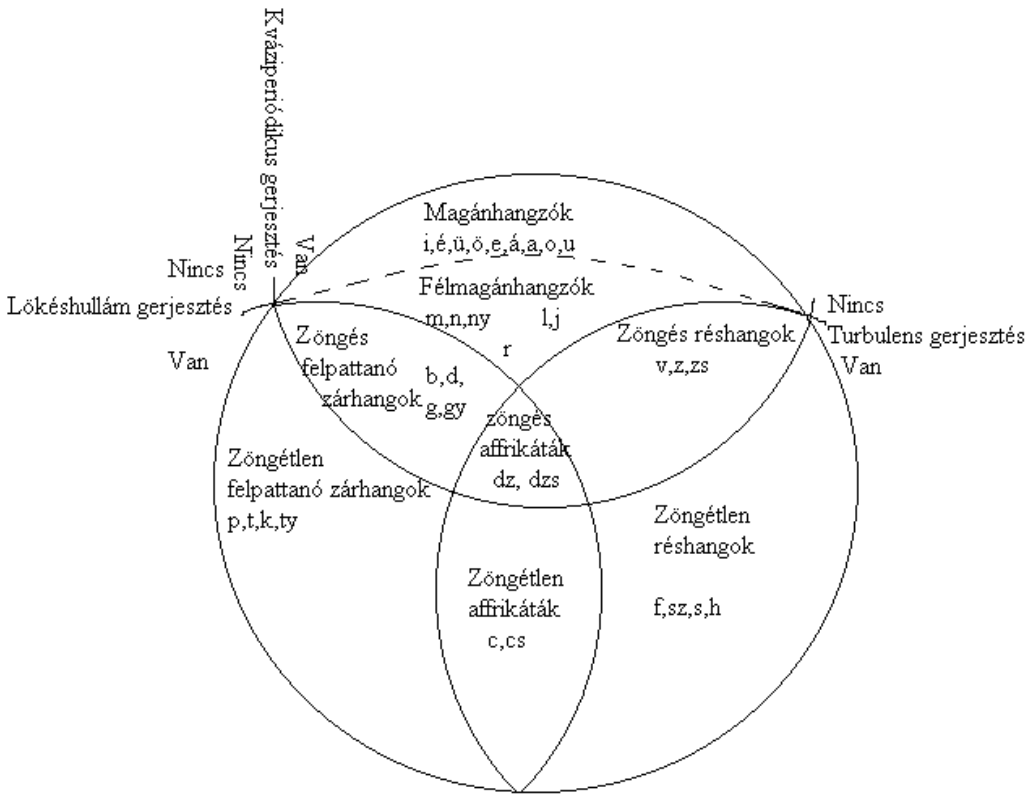
A tudatos hangképzésre a következő tényezők vannak hatással:

- A rezgési frekvencia értéke. Ha alacsony frekvencián rezegtetjük hangszalagjainkat, akkor a rezgés amplitúdója nagyobb, mint ellenkező esetben.
- A hangerő nagysága. Minél nagyobb hangerővel beszélünk, annál gazdagabb a zönge felharmonikusakban, vagyis annál teltebb a hang is hangösszetevőkben.

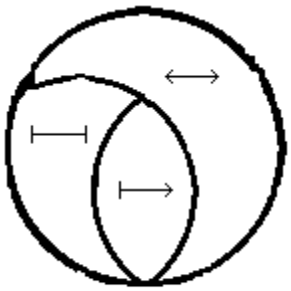
Látható tehát, hogy a hangszalagrezgés minőségét számos tényező befolyásolja, és hogy maga a hangszalagrezgés lefolyása hat a beszédjel minőségére is. Ezért van az, hogy egyes személyek hangja érdes, másoké kellemes hangzású, megint másoké fátyolos, tompa, a gyermekek hangja magas, a nőké általában közepes hangfekvésű, a férfiaké döntően mély. Ha a hangszalagok működését valami akadályozza vagy zavarja (meghűlés, kóros elváltozás, idegi zavar stb.), akkor ez a beszéd minőségében is tükröződik (rekedtség dysphonia). Ha a hangszalagok működése valamilyen okból teljesen megszűnik, akkor kapjuk suttogott beszédet, ami a beszédképződés speciális esete.

Tisztán zörejes gerjesztésnél a hangszalagok nem rezegnek, nyitott állapotot vesznek fel. Ilyenkor a levegő a tüdőből szabadon áramlik a gége feletti üregekbe, és attól függően, hogy az áramlási sebessége mekkora, valamint, hogy az áramlás során milyen akadályokba ütközik, a turbulencia következtében különböző erősségű és hangszínezetű zörejhang keletkezik. Kevert típusú gerjesztésnél a hangszalagok rezegnek, de ugyanakkor a szájüregbe kiáramló pulzáló levegő akadályba is ütközik (pl. a z, zs hangoknál), ami azt eredményezi, hogy a beszédrezgés végeredménye egy zöngés-zörejes hang lesz.” [6]

6. Ábra: Magyar fonéma készlet



7. Ábra: A magyar fonémakészlet flexibilitása



⟷ A hangot nyújthatjuk, csonkíthatjuk
⟼ A hangot csak nyújthatjuk, de nem csonkíthatjuk
⟼⟻ A hangot se nem nyújthatjuk, se nem csonkíthatjuk

Továbbá vizsgálhatjuk a hangokat csonkíthatóság-nyújthatóság szerint, mint ahogy azt a 7. Ábra is mutatja.

Az időfüggvény zöme kvázi periodikus jel. Vizsgálatának módszere spektrálanalízis. Tekintjük a spektrumot, a spektrum maximumai a formánsok, melynek jellemzői:

1. Formáns frekvencia (maximum hely)
2. Formáns szélesség ez az a
3. Formáns intenzitás (maximum érték)

Számítsuk ki, hogy ha az intenzitás aránya 2, akkor mennyi a dB arány.

$$\frac{I_1}{I_2} = 2 \quad 10\log 2 \approx 3$$

4 Jelfeldolgozás

4.1 Mintavételezés

Legyen $x_a(t) \in R \quad t \in R$ egy analóg jel. Ebből a jelből egy analóg/digitális (A/D) átalakító (ADC) segítségével egy *x(n)* digitális jelet állítunk elő, ahol *x(n)* és *n* is diszkrét. Az ADC-nél van egy mintavételezés, megadott időközönként mintát vesz a jelből, és ezeket megtartja. A meghatározott intervallumokhoz diszkrét értékeket rendel. Fontos a kvantálás szerepe, ami lényegében nem más, mint az analóg jelből diszkrét jel létrehozása (valós jelből egész jel). Ez lehet lineáris és logaritmikus.

A hangdigitalizálás éppúgy, mint amikor állóképek gyors egymás utáni lejátszásával mozgás látszatát keltjük, a hang-mintavételező, más szóval digitalizáló eljárás igen rövid időközönként "pillanatfelvételeket" készít (mintát vesz) a rögzíteni kívánt hanganyagról, amelyeket gyors egymásutánban lejátszva rekonstruálhatjuk a felvételt. A hangdigitalizálás többnyire egy mikrofon ,és egy szoftver segítségével történik. A hangdigitalizálás során az analóg hanginformáció digitális jelekké alakul át. Egyes kártyák 48 kilohertzes hang mintavételezésre képesek, ezt azonban csak műszerekkel lehet kimutatni, füllel való érzékelésük szinte lehetetlen. A mintavétel felbontása a hordozott információt leíró bitek számát jelenti.

A mintavételezést szokták sampling-nek is nevezni. A jelfeldolgozás pontossága miatt szokás túl mintavételezni (over sampling) Például a 8-szoros mintavételezés azt jelenti, hogy egy jel helyett nyolcat kapunk és ebből átlagolva kapom meg a tényleges jelet.

Egyik előforduló, elkerülhetetlen probléma a jelfeldolgozás során a kerekítési hiba, vagy kvantálási zaj. Ami abból adódig, hogy két különböző, de egy intervallumba eső valós értékhez is ugyanazt a diszkrét értéket rendeljük hozzá.

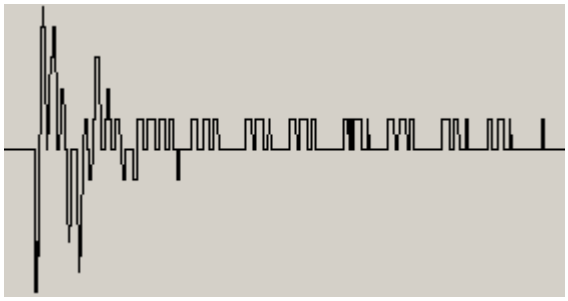
4.2 Használt diagramok

Amennyiben rendelkezünk egy mintával, akkor azt különböző diagramokkal grafikusan is megjeleníthetjük. Így láthatóvá tesszük azt, ami hallható. Számtalan diagramfajta létezik. A dolgozatomban csak néhányat említek meg közölük.

Az egyik legegyszerűbb a hullámforma diagram. Mivel a mai világban elég sok hang formátum terjedt el, melyek struktúrája elég összetett lehet, így ez is kihívást jelenthet. Ez az

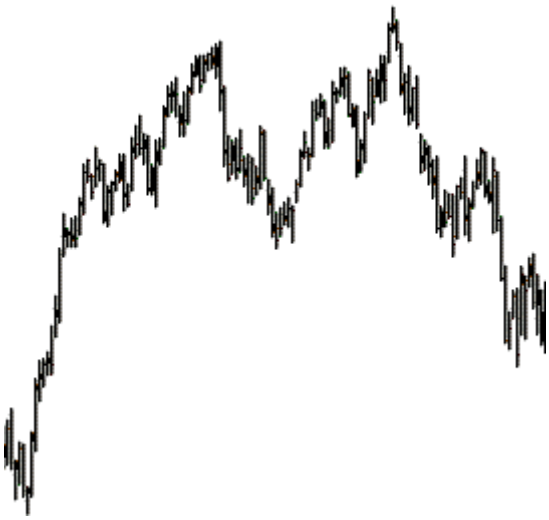
egyszerű idő-impulzus diagram okozhat meglepetéseket. Elég csak a több csatornás tárolásra gondolunk. Nem mindegy az sem, hogy hány bit ír le egy adatot.

8. Ábra: hullámforma diagram.



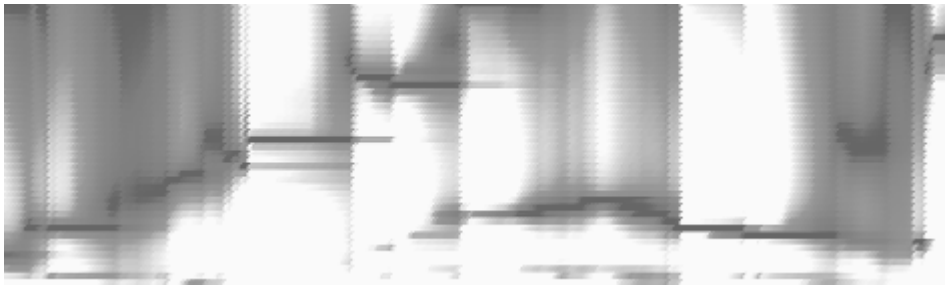
Amennyiben a jel frekvencia összetevőire vagyunk kíváncsiak, abban az esetben a jel spektrumát jelenítjük meg. Ezen a két dimenzió diagramon az egyik tengely a frekvenciákat, a másik az intenzitást reprezentálja. A frekvencia összetevők meghatározására több módszer is létezik, mint például a Fourier transzformációk vagy a lineáris predikció.

9. Ábra: Egy jel spektruma.



A hangot az idő függvényében tekintjük. Amennyiben a frekvencia összetevőket is időfüggvényben szeretnénk megjeleníteni, akkor spektrogramot szoktunk használni. Ez egy háromdimenziós függvény, viszont gyakran két dimenzióban ábrázoljuk. Ekkor az egyik tengely az időt, a másik a frekvencia összetevőket ábrázolja. Az intenzitást az adott pont színével, árnyalatával jelenítjük meg. Amennyiben a kép szürke skálás, úgy minél sötétebb egy pont, annál nagyobb az általa meghatározott frekvencia intenzitása az általa adott pillanatban.

10. Ábra: Egy jel spektrogramja.



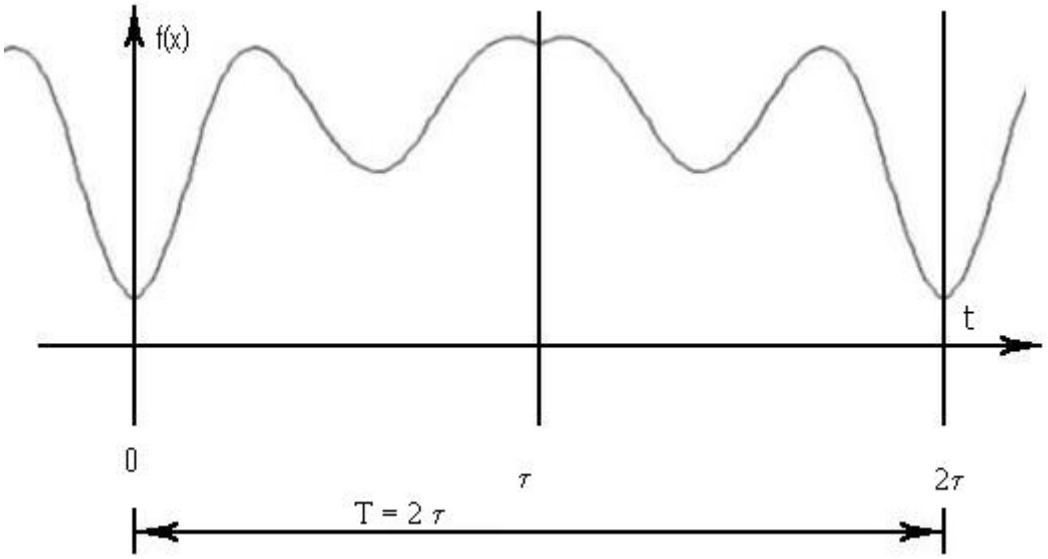
4.3 Fourier és koszinusz transzformáció

Joseph Fourier (1768–1830) francia matematikus felfedezte, hogy minden periodikus függvény felírható szinuszos és koszinuszos jelek összegeként. Ezt jól tudjuk hasznosítani a hangfeldolgozás során. Hasonló elven a kvázi periodikus függvényeket (jeleket), mint például a beszéd, is jól tudjuk közelíteni. A szinuszos és koszinuszos jeleket egy komplex exponenciális függvényként is felfoghatjuk, így a Fourier transzformáció egy komplex függvényt ad eredményül. Az esetek egy részében nekünk elég csak a valós részével foglalkozni.

A koszinusz transzformáció hasonló a Fourier transzformációhoz, viszont ebben az esetben csak a koszinusz függvénnyel foglalkozunk, így az eredmény is valós számok halmazán értelmezhető lesz. Sajátossága még a nagy tömörítési hatásfok, mivel a jelek alacsonyfrekvenciás összetevőire koncentrál.

Mivel a koszinusz egy páros függvény, így ezzel a módszerrel páros függvényeket tudunk közelíteni. Ez azonban nem nagy megszorítás. Egy-egy transzformáció során a jel ablakba eső részére koncentrálunk, és az ablakon kívüli rész nem jelentős a számunkra. Nyugodtan feltételezhetjük azt, hogy azon kívül a jel tükrözve ismétlődik, ahogy az alábbi ábrán látható.

11. Ábra: Egy függvény feltételezett folytatása koszinusz transzformációnál

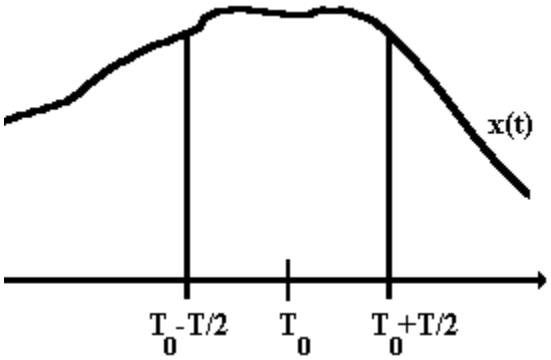


Ezek matematikai megvalósításról a Matematikai háttér című fejezetben olvashatunk.

4.4 Ablakozás

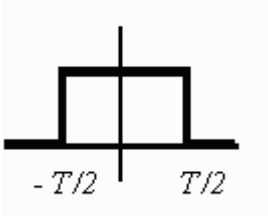
Mivel csak kvázi periodikus jellel és nem periodikus jellel dolgozunk, néhány algoritmus például a diszkrét Fourier transzformáció (DFT) előtt ablakot kell alkalmazni. Az ablak időben leszűkíti az adott jelet. Csak egy meghatározott intervallumon belül tekintem a függvényt, azon kívül úgy veszem, mintha 0 lenne.

12. Ábra: Ablak alkalmazása.



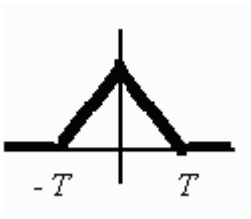
A legegyszerűbb ablak a téglalap ablak, melynek alkalmazás során az adott intervallumból veszem a jeleket. Hátránya, hogy alkalmazása során a kapott spektrum szétkent lesz.

13. Ábra: Téglalap ablak



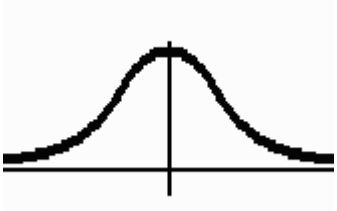
Mivel igazából a jel egy adott időpontban vett spektrumát akarom meghatározni, így az adott időponton kívül eső jeleket kisebb súllyal kell tekintenem, hogy ne torzítsák el a kapott eredményt. Erre a célra alkalmasabb a háromszög ablak.

14. Ábra: Háromszög ablak.



A Hamming ablak azonban a spektrumot "élesíti", mert szűri a spektrumot.

15. Ábra: Hamming-ablak

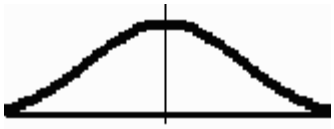


A Blackman-ablak használata esetén majdnem 60dB az eltérés a fő- és a melléknyaláb között.

16. Ábra: Blackman-ablak

Használatos még a Hann ablak, ami leginkább a hamming ablakohoz hasonlít.

17. Ábra: Hann-ablak



5 A szegmentációról általában

A nyelvészet, főleg a fonetika és fonológia egyik kérdése, hogyan lehet a természetes nyelvi beszéd folyamot diszkrét részekre, szegmentekre bontani. Ezek a szegmentek önálló egyedi egységek úgy, mint például a mássalhangzók és a magánhangzók fonetikai szinten, valamint a szavak és a szótagok lexikai szinten. Szegmentáció kapcsán beszélünk emberi és gépi valamint fonetikai és lexikai szegmentációról. Ennek a dolgozatnak a témájában főként a gépi fonetikai szegmentáció tartozik.

A beszéd szegmentáció többek közt egy fontos részproblémája a beszédfelismerésnek, amivel szoros kapcsolatban van. A szegmentálás a beszédfelismerés folyamán tovább finomítható. Mint a legtöbb természetes nyelvi feladat során itt is figyelembe kell vennünk a szövegkörnyezetet, a nyelvtant és a szemantikát is. Ezek ellenére az eredmény elég esetleges, inkább ajánlásnak tekinthető, mint konkrét eredménynek.

5.1 Szegmentálás nyelvészeti háttere

A beszéd szegmentálás legalsó szintje a beszédfolyam fonéma sorrá történő tördelése. A bonyolultsága ennek a feladatnak többek között a beszédhangok kiejtéséből fakad. Az egymás mellett álló hangok számos módon hathatnak egymásra: átmeneteket képezhetnek, összeolvadhatnak vagy akár el is tűnhetnek. Ezek a jelenségek ugyanúgy előfordulnak szomszédos szavak között, mint egy szón belül.

Az az elképzelés, hogy a beszédben az írott szöveghez hasonlóan különböző magánhangzók és mássalhangzók követik egymást az írásbeliség öröksége. Valójában a magánhangzók képzése a környező mássalhangzóktól, a mássalhangzók képzése, a környező magánhangzóktól függ. Például a magyar nyelvben, néhány esetben, a szó eleji mássalhangzó magánhangzó pár esetén a magánhangzó már hat a mássalhangzónak a magánhangzóhoz kapcsolódó részére, tehát a mássalhangzóképzés utolsó fázisára. A b, d, g hangok zárfelpattanásában a második formánsnak megfelelő elem, valamint a v, zs, l, r hangok második formánsa mozog a hangot követő magánhangzó típusának függvényében.

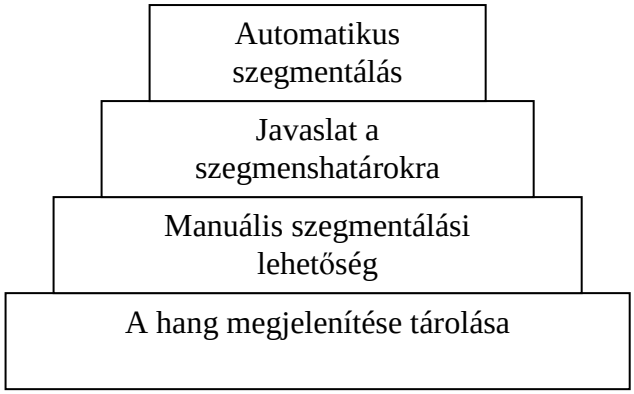
A fent leírtak alapján a statisztikai modellek használhatóak a szegmentáláshoz, és a szavakkal, fonémákkal való címkézésekhez.

Morfológia szinten talán egyszerűbbnek tűnik a szegmentálás. Úgy gondolnák, hogy a beszédközben tartott rövid szünetek leegyszerűsíthetik a szegmens határok beazonosítását, viszont a morfológia kapcsán már különböző grammatikai szabályok is szóba jöhetnek.
A szegmensek meghatározása kézi munkával elég fáradságos. Az így végzett munka minősége eléggé ingadozó lehet, mivel az ember nem képes mindig ugyanúgy odafigyelni a hallott szövegre. Minden segítség sokat jelenthet ebben a munkafolyamatban

5.2 Számítógépes támogatás

A számítógépes támogatásnak több szintje létezik, ahogy az alábbi ábrán is látható.

18. Ábra: a számítógépes támogatás szintjei



A legalapvetőbb a hang tárolás és megjelenítése, amely több formában is lehetséges. A legelterjedtebb állományformák a következők:

1. .VOC: Manapság nem annyira elterjedt, az IBM PC kompatibilis gépek hangkártyái használták.
2. .VAW (WAVeform audio): A Windows operációs rendszerek egyik alap hangformátuma. Tömörítetlen állapotban, RIFF struktúrában tárolja a hangot.
3. .WMA (Windows Media Audio): A Microsoft saját zenei formátuma, ami jogdíjköteles tömörítést tartalmaz.
4. .AU: A Next és a SUN számítógépek operációs rendszerének hangformátuma. A java osztály szinten kezeli.
5. .AIF/.AIFF (Audio Interchange File Format), .AIFC(AIFF-C File Format): Az Apple számítógépek hangformátuma.
6. .MOD, .MID/(MIDI): Egyes hangszereket vezérlô utasításokat tartalmazó hangformátum. Emiatt az állományok mérete lényegesen kisebb, mint más esetekben.

7. .MPA, .MPA2: MPEG videóhoz kapcsoltan előforduló hangformátumok.
8. .MP3: Napjaink egyik legelterjedtebb veszteséges tömörítéssel rendelkező hangformátuma.
9. .RA/.RAM (Realtime Audio): Valamely interneten keresztül továbbított hangfolyam (például rádióműsor) hangformátuma.

Ezek között vannak, amelyek szerkezete egyszerűbb, de olyan is, melyek transzformált, kódolt formában tartalmazzák az adatokat. Egy állomány egy vagy több csatornát is tárolhat. Az adatok hossza is változhat, lehet például 8, 16, 32 bites.

Amennyiben sikerül feldolgoznunk egy állomány formátumot, akkor meg tudjuk jeleníteni az abban tárolt jelet. Néhány elterjedt diagramról már korábban eset szó. Maga a hang megjelenítése is jelenthet segítséget, de nem sokat, ha igazából más eszközzel kell elvégezni a szegmentálást. Jó, ha manuális szegmentálásra is lehetőséget ad egy alkalmazás.

A teljesen automatikus szegmentálás igazából csak egy vízió, de a már meglévő alkalmazások elég jó eredményt szolgáltatnak, a további algoritmusokhoz, mint például a beszédfelismeréshez. Mindenesetre, ha javaslatot tudunk adni a szegmens határokra, ami később finomítható, az már nagy segítséget jelent.

5.3 Javaslatadás a szegmenshatárokra.

Amennyiben a szegmens határokra akarunk egy előzetes kalkulációt a következő lépések segítségével történhet:

1. Egy hang file beolvasása
2. Parametrizálás (MFCC)
3. Paraméter szám csökkentése (Főkomponens analízis)
4. Statisztikai tanuló algoritmus alkalmazása (HMM, súlyozott automata)

A hang file feldolgozásáról már ejtettünk néhány szót. A továbbiakban az azt követő lépésekkel fogunk foglalkozni.

5.3.1 Mel-frekvenciás kepsztrális komponens (MFCC)

A Mel-frekvenciás kepsztrális komponens az angol nyelvű irodalom Mel-frequency cepstral coefficients vagy röviden MFCC néven használja.

A hangfeldolgozásban a Mel-frekvenciás kepsztrum (mel-frequenc cepstrum MFC) a hang rövididejű energia spektrumát reprezentálja. Az MFC egy lineáris koszinusz transzformáció

alapszik. A hang egyfajta kepsztrális megjelenítéséből ered (spektrum spektruma). A különbség a kepsztrum és a mel-frekvenciás kepsztrum között, hogy ez utóbbinál a frekvenciák mel-skáláját vesszük alapul, ami jobban illeszkedik az emberi halláshoz. Így ez utóbbi jobban leírja a beszédet például a hangok tömörítése során. Az MFCC egyik nagy hátránya, hogy nagyon érzékeny a zajokra, ezért léteznek finomításai, melyek megpróbálják ezt a hátrány kiküszöbölni.

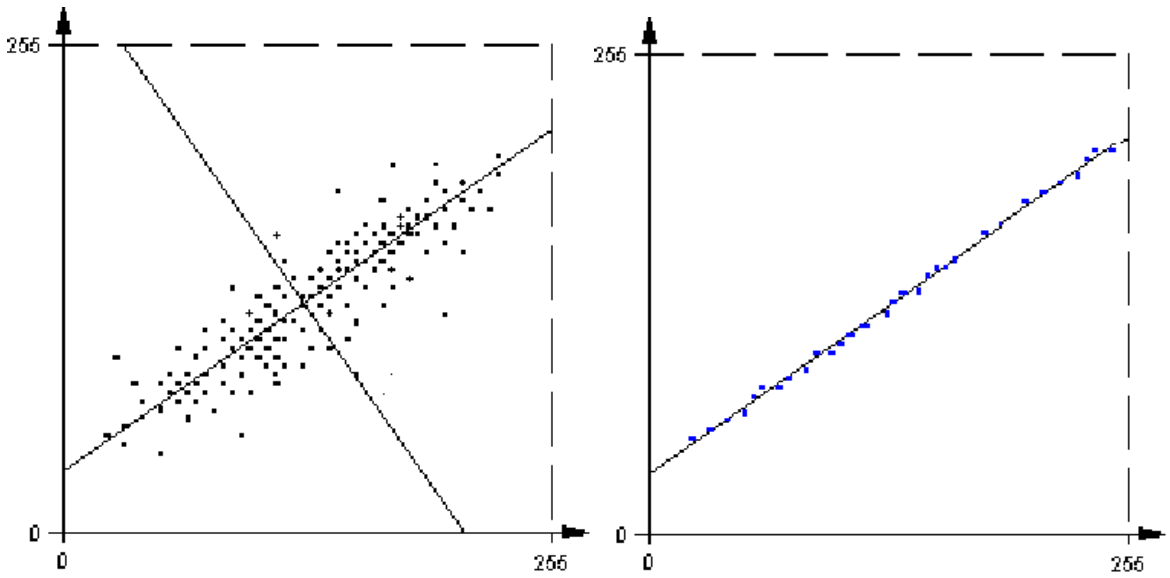
MFCC meghatározása általában a következő lépésekből áll:

1. Elkészítjük a jel Fourier transzformáltját ablak alkalmazásával.
2. Átváltjuk a kapott frekvenciákat Mel-skálára.
3. Minden egyes Mel frekvenciának vesszük a logaritmusát.
4. Az így kapott értékek diszkrét koszinusz transzformáltját vesszük.
5. A kapott spektrum amplitúdói az MFCC együtthatók.

5.3.2 Főkomponens analízis (Principal Component Analysis)

A főkomponens analízis egy koordináta transzformáció. Az eljárás során megpróbáljuk a több dimenziós adathalmazt oly módon transzformálni, hogy a nagyobb mértékben meghatározó komponensek nagyobb intervallumból, a kisebb mértékben meghatározók kisebb intervallumból vegyék fel értékeiket. Az alábbi ábra bal oldali felén láthatunk egy vízszintes, függőleges koordináta rendszerben ábrázolt adathalmazt illetve annak transzformációját. A transzformáció után a meghatározó komponens a bal alsó sarokból a jobb felsőbe vezető egyenes. Ezen mentén jobban „szóródnak” az értékek, mint a rá merőleges egyenes mentén.
Előfordulhat olyan eset is, hogy a koordináta transzformáció után kevesebb értékkel leírhatjuk az adathalmazt, ahogy az alábbi ábra jobb oldali felén láthatjuk. Ebben az esetben a két paraméterrel meghatározott pontok, egy jobb alsó sarokból bal felső sarokba vezető egyenesre illeszkednek, így ezen egyenesen elfoglalt helyzetükkel jellemezhetőek.

19. Ábra: Példák főkomponens analízisre



6 Matematikai háttér

6.1 Fourier és koszinusz transzformáció

Fourier sor azt mondja meg, hogy egy $[a,b]$ intervallumon definiált függvény, hogy állítható elő $f(t)=\sum_{k=-\infty}^{\infty} c_k \exp(ikt)$ alakban (adott frekvenciájú értékekkel), vagyis, hogy a függvény hogyan állítható elő különböző frekvenciájú szinuszos alakban.

Inverz Fourier transzformáció azt mondja meg, hogy a megadott frekvenciájú összetevőkből hogyan állítható vissza az eredeti jel.

$$x(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty} X(f)\exp(i2\pi ft)dt \quad t\in(-\infty,\infty) \tag{1}$$

Egy $c(f)\cos(f2\pi t+\varphi)$ alakú jel jön ki.

$$X(f)=\int_{-\infty}^{\infty} x(t)\exp(-i2\pi ft)dt \quad f\in(-\infty,\infty) \tag{2}$$

$X(f)$ az $x(t)$ Fourier transzformáltja vagy spektruma. Spektrum: minden olyan függvény, ami megadja, hogy adott frekvenciájú függvényekből mennyit kell venni, és mennyire kell eltolni.

6.1.1 A Fourier Transzformáció tulajdonságai

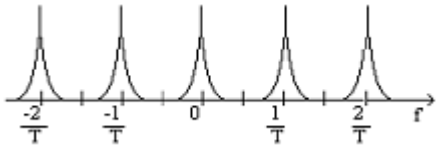
(az X -et jelölhetjük így is $X=F(x)$):

- $F(x*y)=F(x)\cdot F(y)$
- $F(x\cdot y)=F(x)*F(y)$

  A konvolúciós szorzást közönséges szorzásba, míg a közönséges szorzást konvolúciós szorzásba viszi át.
- $F(x')=i2\pi fF(x)$
- $F(x^*)=T(F(x))^*$
- $F(x')=i2\pi fF(x)$
- $F(x^*)=T(F(x))^*$, ahol *T*: tükrözés *f=0*-ra.

- $F(x_p) = \sum_{k=-\infty}^{\infty} c_k \delta(f - k/T)$, ahol c_k az x_p T periódusú $\frac{1}{T}$ lépésközű diszkrét függvény Fourier együtthatója (ennek csak általánosítva létezik Fourier transzformáltja), mely mindenütt *0* kivéve az $\frac{n}{T}$ pontokban

20. Ábra: $\frac{1}{T}$ lépésközű diszkrét függvény



- $F(x) \cdot F(x)^* = F(x * T(x^*))$, ahol x egy komplex szám. Ha *x* valós, akkor $x=x^*$ és $F(x * T(x^*)) = F(\int_{-\infty}^{\infty} x(\tau) \cdot x(t+\tau) d\tau$, ami nem más mint x autókorrelációs függvénye.
- Ha *x* valós, akkor $x=x^*$ és $F(x)=T(F(x))^*$

  F(x) Valós része páros függvény

  Képzetes része páratlan függvény
- *T* periódusú függvény *F* transzformáltja $\frac{1}{T}$ közű diszkrét függvény. Δt közű diszkrét függvény *F* transzformáltja $\frac{1}{\Delta t}$ periódusú periodikus függvény. Δt közű diszkrét $n \cdot \Delta t$ (*n* darab mérési eredményem van) periódusú periodikus függvény *F* transzformáltja $\frac{1}{\Delta t}$ periódusú periodikus, $\frac{1}{n\Delta t}$ közű diszkrét függvény. Ez a diszkrét Fourier transzformált, *n* értékből *n* értéket állít elő.

6.1.2 A Diszkrét Fourier Transzformáció (DFT) képlete:

Diszkrét esetben a Fourier Transzformációban szereplő integrált egyszerű szummával helyettesíthetjük.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i 2\pi k n / N} \quad k = 0,1,\cdots,N-1 \qquad (3)$$

6.1.3 Gyors Fourier Transzfomáció (FFT):

Gyors Fourier Transzformációra Cooley-Tuckey algoritmusa (1965) szolgál. Kihasználjuk, hogy a lassú szorzások több esetben is gyors összeadásokkal helyettesíthetők.

Legyen $N=2^m$, $w = e^{-i2\pi/N}$

$k=k_{m-1}2^{m-1}+ \ldots +k_0$ $n=n_{m-1}2^{m-1}+ \ldots +n_0$ (bináris felbontás) $k_i, n_j = 0$ vagy 1 $\forall i, j \in [0, m-1]$-re

Ha $i+j \geq m$, akkor $w^{k_i n_j 2^{i+j}} = w^{k_i n_j 2^m 2^{i+j-m}} = (w^N)^{k_i n_j 2^{i+j-m}} = 1$ (4)

$$X(k_{m-1}, k_{m-2}, k_0) = \sum_{n_0=0}^{0} \cdots \sum_{n_{m-1}=0}^{1} x(n_{m-1}, \ldots, n_0) w^{\sum_{i,j=0}^{m-1} k_i n_j 2^{i+j}} =$$

$$\sum_{n_0=0}^{0} \cdots \sum_{n_{m-1}=0}^{1} x(n_{m-1}, \ldots, n_0) w^{\sum_{i=0}^{m-1} k_i 2^i 2^{m-1} n_{m-1}} \cdot w^{\sum_{i=0}^{m-1} k_i 2^i 2^{m-2} n_{m-2}} \cdots w^{\sum_{i=0}^{m-1} k_i 2^i 2^0 n_0} =$$

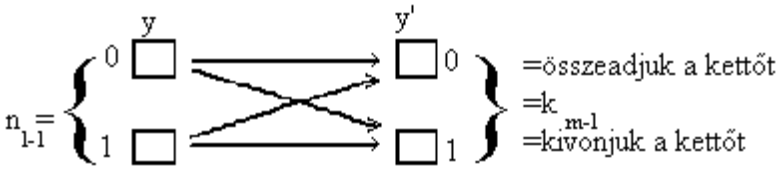
$$\sum_{n_0=0}^{1} (\ldots (\sum_{n_{m-1}=0}^{1} x(n_{m-1}, \cdots, n_0) w^{\sum_{i=0}^{0} k_i 2^i 2^{m-1} n_{m-1}}) \cdots) w^{\sum_{i=0}^{m-1} k_i 2^i 2^0 n_0} \quad (5)$$

$$\sum_{i,j=0}^{m-1} k_i n_j 2^{i+j} = \sum_{j=m-1}^{0} \sum_{i=0}^{m-1-j} k_i n_j 2^i 2^j \quad (6)$$

Feltesszük, hogy n_l-re már elvégeztük minden lehetséges k_i mellett az összegzéseket. Vegyük észre, hogy az eredmények csak $n_{l-1},\ldots,n_0$ $k_0,\ldots,k_{m-1-l}$ értékektől függnek. Egy segédváltozót vezetünk be: $y(\underbrace{k_0,\ldots,k_{m-1-l}}_{\text{m-l db bit}}, \underbrace{n_{l-1},\ldots,n_0}_{\text{l db bit}})$ a részeredmények tárolására mindig elegendő m db elem.

$$y(k_0,\ldots,k_{m-1-l}, k_{m-l}, n_{l-2},\ldots,n_0) := \sum_{n_{l-1}=0}^{1} y(k_0,\ldots,k_{m-1-l}, \underline{n_{l-1}}, n_{l-2},\ldots,n_0) w^{(\sum_{i=0}^{m-l} k_i 2^i) n_{l-1} 2^{l-1}} \quad (7)$$

21. Ábra: Lepke művelet



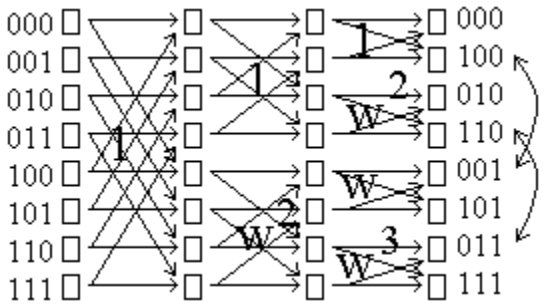
Ha $n_{l-1}=1$ és $k_{m-l}=0$, akkor a művelet eredménye $\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}$. Mivel $n_{l-1}=1$, ezért ezt nem kell kiírni, és azért m-l-1-ig összegzünk csak, mivel $k_{m-l}=0$ és ezt kiejtjük

Ha $n_{l-1}=1$ és $k_{m-l}=1$, akkor a művelet eredménye:

$$w^{\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}} w^{2^{m-1}} = w^{\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}} w^{\frac{N}{2}} = w^{\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}} \exp\left(-i2\pi / N^{\frac{N}{2}}\right) = w^{\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}} \exp(-i\pi) = -w^{\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}} \quad (8)$$

Pl m=3

22. Ábra: Gyors Fourier Transzformáció *m=3* esetben



Legyen kezdetben *y=x.* Az **22. Ábrán** baloldalt találhatóak az indexek binárisan, jobb oldalt az első oszlopbeli vektor diszkrét fourier transzformáltját kapjuk meg fordított indexeléssel, így helycseréket kell végrehajtani. Először azokra az elemekre hajtjuk végre a lepkeműveletet, melyek az első indexben eltérnek, az első transzformáció *m-l*=0, *l*=3 esetén történik. Sorban végrehajtjuk a második, harmadik indexben eltérő elemekre is (*m-l*=1 *l*=2, *m-l*=2 *l*=1). Ha 3-*l*=0, akkor $q = w^{\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}}$, és üres szumma definíció szerint 0, ezért w^0=1, így q=1. Ha *m-l*=1, akkor ha k_0=0, $q = w^{\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}} = 1$; ha k_0=1, $q = w^{\sum_{i=0}^{m-l-1} k_i 2^{i+l-1}} = w^2$. $w^{\sum_{i=0}^{1} k_i 2^i}$ Számolási táblázatot kapunk így. Ez egy diszkrét Fourier transzformációt valósít meg.

6.1.4 Diszkrét Koszinusz transzformáció (DCT)

A diszkért koszinusz transzformáció egy a Fourier transzformációhoz nagyon hasonló, lineáris, invertálható N-edfokú valós függvény $F : R^N \to R^N$. Felfogható egy NxN-es invertálható négyzetes mátrixként is. Több egymáshoz nagyon hasonlító képlet is létezik a Diszkrét Koszinusz transzformáció. A következő képletek mind $x_0, x_1, ..., x_{N-1}$ N valós számot transzformálnak $X_0, X_1, ..., X_{N-1}$ valós számra.

I. DCT-I

$$X_k = \frac{1}{2}(x_0 + (-1)^k x_{N-1}) + \sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi}{N-1} nk\right] \qquad k = 0, ..., N-1$$

II. DCT-II

$$X_k = \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N-1} nk\right] \qquad k = 0,\ldots, N-1$$

III. DCT-III

$$X_k = \frac{1}{2}x_0 + \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N} n\left(k+\frac{1}{2}\right)\right] \qquad k = 0,\ldots,N-1$$

IV. DCT-IV

$$X_k = \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right] \qquad k = 0,\ldots,N-1$$

6.2 Hertz - Mel konverzió képletei:

A képletekben az f a Hertz az m a Mel skálán vett értéket jelzi.

$$m = 1127.01048 * \log_e(1 + f / 700)$$

$$f = 700 * (e^{m/1127.01048} - 1)$$

A következő alternatív képletek is ismertek:

$$m = 2595 * \lg(1 + f / 700)$$

$$m = (1000 / \log_{10} 2) * \log_{10}(1 + f / 1000)$$

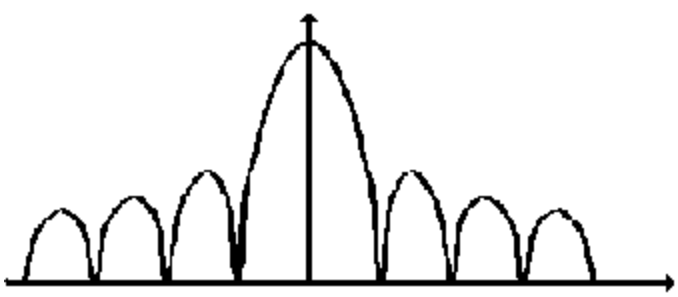
6.3 Ablakok

Legyen w*(t)* ablak függvény. A $w(t-T_0)$ függvény az *x(t)* jelet képzi le a következő módon: $x(t)w(t-T_0)$,ahol *x(t)* az eredeti jel.

A w(t) téglalap ablak értéke konstans 1 az intervallumban.

*X(f) (X*W)(f)* ablakozás a spektrumon $W = \frac{\sin f}{f}$

23. Ábra: $\left|\frac{\sin f}{f}\right|$ **képe**



Ezzel veszem a *X(f)* konvolúciós szorzatát (súlyozott átlagát veszem az *f* frekvenciákon a súly az $|W|$) cél amellékpúpok kisebbek legyenek.

Háromszögablak: ilyenkor a melléknyaláb 25 dB-el kisebb. Háromszög ablak értéke 0-nál 1, és az intervalum szélei felé lineárisan csökken.

Hann ablak: Ez a cos*(t)* egy szelete 0,5cos*(t)*+0,5

Hamming-ablak: Egy picikét emelünk a Hann ablakon, 0,46cos*(t)*+0,54

Blackman-ablak: $a_0+a_1\cos(t)+a_2\cos(2t)$ Az $a_2\cos(2t)$ miatt a függvény leér a vízszintes tengelyén. a_0=0,42 a_1=0,25 a_2=0,04

6.4 Rejtett Markov modell

A háttérben van egy állapottér, aminek elemei $\{S_1,\ldots,S_N\}$

y kimenet az állapotok függvénye $y = y(s)$

$M = (A, B, \pi)$ (markov-modell)

$\pi = (\pi_1,\ldots,\pi_N)$ valószínűségi vektor (π_i-k valószínűségek $\sum \pi_i = 1$) az állapotok kezdeti valószínűsége $\pi_i = p$ (az első pillanatban az állapot S_i)

$A = (a_{ij})$ $1 \le i \le j \le N$ állapotátmenetek valószínűségeinek mátrixa $a_{ij} = p$ (az adott pillanatban az aktuális állapot S_j | az előző pillanatban az állapot S_i volt)

$B = (b_1,\ldots,b_N)$ vektor, ahol b_i feltételes eloszlásfüggvény, a kimenő jelek eloszlás függvénye feltéve, hogy az aktuális állapot S_i. A kimenő jel nem függ az időtől csak az aktuális állapottól $b_i = \sum_{m=1}^{M} c_m \mathrm{N}(\mu_m, \sigma_m)$ normális eloszlások keveréke

Tekintsünk egy megfigyelési sorozatot

$Y_T = [y_1,\ldots,y_T]$ konkrét output $Q = [q_1,\ldots q_T]$ állapot sorozat

Mi annak a valószínűsége, hogy ez jön ki, feltéve, hogy a modell ismert?

$P = p(Y_T, M) = \sum_Q p(Y_T|Q,M)p(Q,M) = \sum_Q \prod_{t=1}^{T} p(y_T|q_T,M)p(Q,M)$ N^T db lehetséges állapotsorozat van. Csak *Q*-tól függnek a kimenetek. Feltesszük, hogy *B* eloszlások diszkrétek

6.4.1 Előre-hátra algoritmus a P kiszámítására:

Előre felé kiszámítás

Legyenek $y_1,\dots y_T$ fix (mért értékek). Definiáljuk az $\alpha_t(i) = p(y_1,\dots,y_n \text{ és } q_t = S_i, M)$ függvényt az $1 \le t \le T$, $1 \le i \le N$ esetben, ahol *t* az időt, az az *i* állapot sorszámot jelöli. Legyen $\alpha_1(i) = \pi_i b_i(y_i)$, és definiáljuk iteratív módon $a_{i+1}(j)$-t.

$$\alpha_{t+1}(j) = \left(\sum_{l=1}^{N} \alpha_t(i) a_{ij}\right) b_j(y_{t+1}) \quad T-1 > t \ge 1 \tag{9}$$

Az a_{ij} jelöli az *i*. állapotból a *j*. állapotba való áttérés valószínűségét, így $P = \sum_{i=1}^{N} \alpha_T(i)$.

Vissza felé kiszámítás esetén, az előre felé kiszámoláshoz hasonlóan definiáljunk egy $\beta_i(j) = p(q_T = S_j \text{ és } y_{t+1},\dots,y_T)$ függvényt az $1 \le t \le T$, $1 \le j \le N$ esetben. Legyen $\beta_{T+1}(j) = 1 \ \forall j = 1,\dots,N$, és definiáljuk iteratív módon $\beta_t(i)$-t.

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} b_j(y_{t+1}) \tag{10}$$

A valószínűséget $a_{i+1}(j)$ és $\beta_t(i)$ felhasználásával a következő módon számolhatjuk ki:

$P = p(y_1,\dots,y_T) = \sum_i \sum_j p(y_1,\dots,y_T \text{ és } q_t = S_i)$ t tetszőleges. Tételezzük fel hogy t.

$t-1$	t	$t+1$	$t+2$
y_{t-1}	y_t	y_{t+1}	y_{t+2}
	S_i	S_j	

pillanatban S_i t+1. pillanatban S_j állapot van érvényben: . Ekkor a

valószínűség kiszámítását a következő módon folytathatjuk:

$p(q_t = S_i \text{ és } q_{t+1} = S_j) p(y_{t+1} | q_{t+1} = S_j) p(q_{t+1} = S_j \text{ és } y_{t+2},\dots y_T)$, ezt összefoglalva az alábbi képletet kapjuk:

$$P = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(y_{t+1}) \beta_{t+1}(j) \quad T-1 \ge t \ge 1 \tag{11}$$

Implementációs megfontolásból α_t helyett $c_t \alpha_t$-vel számolhatunk így a számlálóban és nevezőben is megjelenik $c_1,\dots,c_T$ ezért ejthetjük.

6.4.2 Újrabecslési formulák

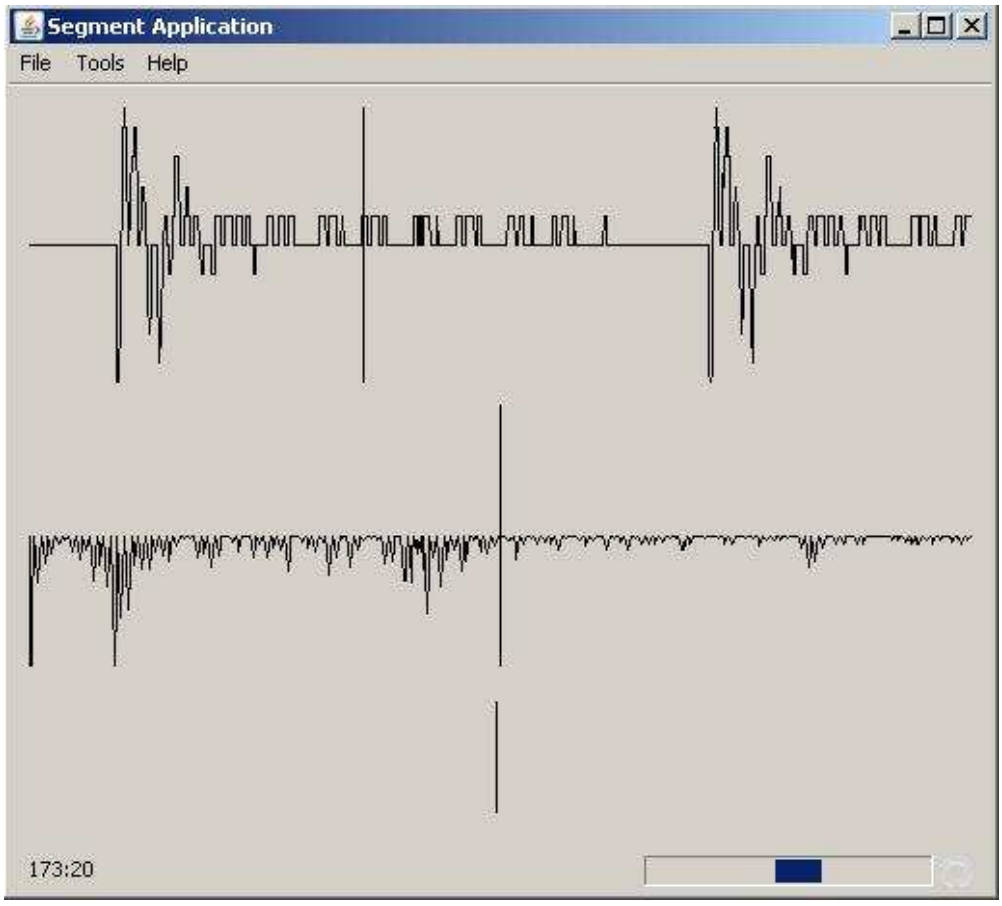
Kiindulunk egy M_0 Markov Modellből ez a kezdeti modell. $y_1^{(k)},\ldots,y_T^{(k)}=Y_t^{(k)}$ a szóról T db mintát veszünk k=0,1,… M_0-ból és $Y_T^{(0)}$-ból újrabecsüljük az M_0 Modellt. M_1-ból és $Y_T^{(1)}$-ból újrabecsüljük az M_2 Modellt és így tovább. S_i-ből S_j-be való átmenetek várható száma (független az időtől): $\gamma_{ij}=\frac{1}{P}\sum_{t=1}^{T-1}\alpha_t(i)a_{ij}b_j(y_{t+1})\beta_{t+1}(j)$. S_i-ből való átmenetek várható száma:

$$\gamma_i=\sum_{j=1}^{N}\gamma_{ij}\quad a'_{ij}=\frac{\gamma_{ij}}{\gamma_i}\quad \pi'_i=\frac{1}{P}\alpha_1(i)\beta_1(i)\quad b_j(y=v_k)=\frac{\sum_{t,y(t)=v_k}\alpha_t(j)\beta_t(j)}{\sum_t\alpha_t(j)\beta_t(j)}$$

7 Konkrét megvalósítás

Az eddig leírtakat egy konkrét megvalósításon keresztül tekinthetjük át a legjobban. Az példa több helyen finomítható, átgondolható, viszont egy kiindulási alapnak tökéletesen megfelel. Fejlesztése Windows XP környezetben történt Java nyelven NetBeans IDE 6.1 fejlesztő eszköz segítségével. A felhasználói felület Swing alapú. Az elkészült program (remélhetőleg) könnyen felhasználható más platformokon is.

24. Ábra: képernyőkép az elkészült alkalmazásról.



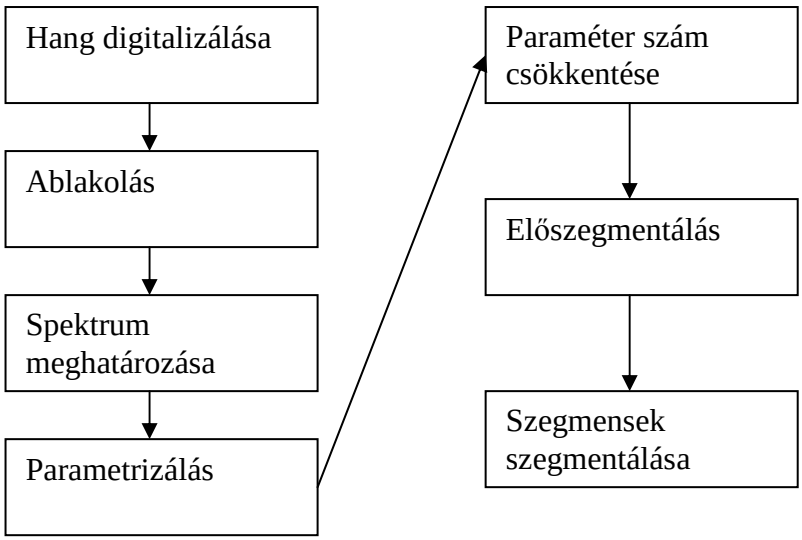
Első körben a hang rögzítését kell megoldanunk. Erre a Windowsos felületeken elterjedt RIFF szerkezetű WAV állományokat fogjuk alkalmazni. A kódolásnál megmaradunk az általános PCM kódolásnál. Az adatokat 16 biten reprezentáljuk. Maga a mintavételezés történhet például Sound Recorderrel, amely a Windows rendszerek része. Az általam elkészített alkalmazás lehetőséget ad ezen állományok beolvasására. A képernyőn az adatállományok első csatornájának hullámformáját láthatjuk. A mintavételezési pontok számát (Sample

Length) az állomány frekvenciájának felére állítjuk, így megpróbáljuk elkerülni a túl mintavételezést. Ezt az értéket a felhasználó a Tool/Options menüben megváltoztathatja.
A hullámforma diagramon mozgatott kurzor pozícióját kiírja az alkalmazás a bal alsó sarokba. Az aktuális időpillanatot egy függőleges vonal is jelzi. A bal egér gomb kattintásával a függőleges vonal által kiválasztott időpontra vonatkozólag vesz egy Hamming ablakot a jelből következő képlettel:

$$h[n] = 0.54 - 0.46\cos\frac{2\pi n}{N}$$

ahol n = 0 ... N-1, és N az ablak mérete. Az így vett mintán Fourier transzformáció segítségével elkészül a frekvencia összetevőinek képe.
Lehetőség van a jel spektogramának elkészítésére is. Ezt a File/Spectogram menüponton keresztül lehet kérni. A spektogram elkészítése sok időt vesz igénybe, épp ezért a háttérben zajlik. A jobb alsó sarokban megjelenő progress bar jelzi, hogy a művelet folyamatban van. Amint befejeződik a feldolgozás megjelenik a kész spektogram zöld árnyalatban, ami 255256 különböző értéket tud megjeleníteni. A spektrum mélységét a tools/options alatt változtathatja meg a felhasználó a „Deep of spectogram” érték megváltoztatásával. A program a kapott intenzitás értékeket megszorozza ezzel az értékkel, majd az így kapott eredményt jeleníti meg. A 255-nél nagyobb értékeket 255-nek veszi.
A tools/options menüpont alatt lehetőség van a „Use Mel Scale”, „Use log of powers” jelölő négyzetek kiválasztására. Amennyiben ezeket megadjuk, úgy a spektrumok és spektogramok megjelenítése során a Mel skálát illetve az intenzitások logaritmusát fogjuk figyelembe venni.
Az MFCC menüpont kiválasztásával előállítjuk a mel-frekvenciás kepsztrális komponensek egy .CSV formátumát. Az így kapott paramétereket elmenthetjük egy szövegállományba a további feldolgozáshoz. A spektogramhoz hasonlóan ez is egy hosszú folyamat, így ez is a háttérben fut.
A további fejlesztésekben a paraméterek számát egy főkomponens analízis segítségével megpróbálom csökkenteni. Az így kapott vektorok alkalmasak lesznek egy Rejtett Markov Modellel elvégzendő szegmentálásra. A kapott szegmensekre újra elvégzek majd egy szegmentálást, remélhetőleg így finomítani tudom a kapott eredményeket. Ez a folyamat látható a következő ábrán is.

25. Ábra: Szegmentálás folyamata.



A program megvalósítása során igyekeztem már meglévő csomagokat alkalmazni. A Fourier transzformációhoz a VisAD FFT osztályát használtam, ami a GNU licensz alá tartozik, így az elkészült alkalmazásra is a GNU előírásai vonatkoznak. A diszkrét koszinusz transzformációra a Developer.com-on megjelent „Understanding the Discrete Cosine Transform in Java” című cikkből vettem a kódot. Ezeket a függelékben mellékeltem. A Rejtett Markov Modelt megvalósítására a Jahmm-ot tervezem használni, ami egy létező JAVA csomag.

A továbbiakban számos lehetőség van a program finomítására. A használt FFT és diszkrét koszinusz transzformáció csomagok az alkalmazásban nagyon lassúak. Ezek kiváltásával több előnyre is szert teszek. A futási idő rövidebb lesz, és mentesülök a GNU licensz megszorításai alól. Végezetül szeretnék felsorolni néhány funkciót, amivel a későbbiekben bővíthető a rendszer:

1. Több állományformátum támogatása
2. Manuális szegmentáció
3. Szegmentációs javaslatok
4. Használt külső objektumok lecserélése
5. Fonémák címkézése
6. Spektrumok összehasonlítása

8 Irodalom jegyzék

[1] Csipak Andrea: A Rejtett Markov modell alkalmazása a beszédfelismerésben. (1996) Szakdolgozat, Debrecen

[2] Dr. Gordos Géza, Takács György: Digitális beszédfeldolgozás. (1983), Műszaki Könyvkiadó, Budapest

[3] Gósy Mária, Siptár Péter: Beszédkutatás Tanulmányok az elméleti és az alkalmazott fonetika köréből. (1993) A Magyar Tudományos Akadémia Nyelvtudományi Intézete, Budapest

[4] Gósy Mária: Beszédkutatás Tanulmányok az elméleti és az alkalmazott fonetika köréből. (1994) A Magyar Tudományos Akadémia Nyelvtudományi Intézete, Budapest

[5] A. Jászó Anna: A Magyar Nyelv Könyve (1991) Trezor kiadó, Budapest

[6] Olaszy Gábor: Elektronikus beszédelőállítás. A magyar beszéd akusztikája és formánsszintézise. (1989), Műszaki Könyvkiadó, Budapest

[7] Pauka Károly: Hallászlélektan a beszédmegértés alaptényezői. (1980) Eötvös Loránd Tudományegyetem, Bölcsészettudományi Kar Tankönyvkiadó, Budapest

[8] Peter M. Ridge, David M. Golden, Ivan Luk, Scott E. Sindorf: Hangkártya Sound Blaster. (1995), Panem, Budapest, Fordította: dr. Nyikos Lajos

[9] Székely Vladimír: Képkorrekció, hanganalízis, térszámítás PC-n. Gyors Fourier Transzformációs módszerek. (1994), ComputerBooks, Budapest

[10] Varga Balázs: A Fourier-analízis alkalmazása a távírótechnika oktatásában. (1988), Szakdolgozat, Debrecen

[11] Angol Wikipedia (Speech segmentation): http://en.wikipedia.org/wiki/Speech_segmentation

[12] Angol Wikipedia (Mel-frequency cepstrum): http://en.wikipedia.org/wiki/Mel_frequency_cepstral_coefficient

[13] Angol Wikipedia (Mel scale): http://en.wikipedia.org/wiki/Mel_scale

[14] Angol Wikipedia (Principal components analysis): http://en.wikipedia.org/wiki/Principal_Component_Analysis

[15] Angol Wikipedia (Hidden Markov model): http://en.wikipedia.org/wiki/Hidden_Markov_model

[16] Angol Wikipedia (Fourier series): http://en.wikipedia.org/wiki/Fourier_series

[17] Angol Wikipedia (Discrete cosine transform):
http://en.wikipedia.org/wiki/Discrete_cosine_transform

[18] Magyar Wikipedia (Diszkrét koszinusz transzformáció):
http://hu.wikipedia.org/wiki/Diszkr%C3%A9t_koszinusz_transzform%C3%A1ci%C3%B3

[19] Jahmm - Hidden Markov Model (HMM):
http://www.run.montefiore.ulg.ac.be/~francois/software/jahmm/

[20] Understanding the Discrete Cosine Transform in Java:
http://www.developer.com/java/other/article.php/3619081

[21] VisAD FFT class: http://www.ssec.wisc.edu/~dglo/docs/visad/math/FFT.html

9 Függelék

9.1 Néhány fontosabb függvényt tartalmazó util osztály kódja

```
package segment;

import javax.sound.sampled.*;
import java.util.*;
//import java.
/**
 * Util for audio process
 * @author Cseh Miklós Zsolt
 */
public class util {

//public static void readSoundFile(String pFileName, LinkedList<byte[]>
pListaudioBytes) {
public static int[][] readSoundFile(String pFileName) {
int totalFramesRead = 0;
int iSample = 0;
int l;
int h;
java.io.File fileSound = new java.io.File(pFileName);
LinkedList<byte[]> listFrame = new LinkedList<byte[]>();
try{
   //javax.sound.sampled.AudioInputStream audioInputStream =
   //   javax.sound.sampled.AudioSystem.getAudioInputStream(fileSound);
   AudioInputStream audioInputStream =
      AudioSystem.getAudioInputStream(fileSound);
   AudioFormat audioFormat = audioInputStream.getFormat();
   int ibytesPerFrame = audioFormat.getFrameSize();
   int iChannels = audioInputStream.getFormat().getChannels();
   int iSampleSize = audioFormat.getSampleSizeInBits();
   SampleLength = (int)Math.pow(2,Math.floor(
           Math.log(audioFormat.getSampleRate())/Math.log(2))+1)/2;
   audioFormat = null;

   int numBytes = 1024 * ibytesPerFrame;
   byte[] paudioBytes = new byte[numBytes];

   int numBytesRead = 0;
   int numFramesRead = 0;
    // Try to read numBytes bytes from the file.
```

```
    while ((numBytesRead =
      audioInputStream.read(paudioBytes)) != -1) {
      //pListaudioBytes.add(paudioBytes);
      listFrame.add(paudioBytes);
      // Calculate the number of frames actually read.
      numFramesRead = numBytesRead / ibytesPerFrame;
      totalFramesRead += numFramesRead;
    }

//Convert raw data to channels
//int iframeLength = (int) audioInputStream.getFrameLength();
//int[][]
if (iSampleSize == 16){
int [][] piReturn = new int[iChannels][totalFramesRead*numBytes/(iChannels*2)];

    for(byte[] framecounter : listFrame){
       for (int iCounter = 0; iCounter < framecounter.length;) {
          for (int channel = 0; channel < iChannels; channel++) {
             l = (int) framecounter[iCounter++];
             h = (int) framecounter[iCounter++];
             piReturn[channel][iSample] = (h << 8) + (l & 0x00ff);
            //16 bit sample
            //TODO to implement other samples
          }
          iSample++;
       }
    }
    return piReturn;
}
else return null;
}
catch(Exception ex)
{
    //TODO to implement
    ex.printStackTrace();
    return null;
}


}

public static double[] HammingWindow(int[] Source, int Start, int Length){
double [] res = new double[Length];
for(int i = 0; i < Length; i++)
```

```
{
    res[i] = 0.54 - 0.46*Math.cos(2*Math.PI*i/Length); //Hamming Window
    res[i] = res[i]*Source[i+Start];
}

return res;
}

public static int[] CalcFFTRel(int[] values, int Start, int Length, int avg,
        int MaxValue){
double[] Window = util.HammingWindow(values, Start, Length);
double[][] FFTSource = new double[2][Window.length];
int Max = 0;


   for(int i = 0;i<Window.length;i++){
      FFTSource[0][i] = Window[i];
      FFTSource[1][i] = 0;
   }

   double[][] FFTRes = FFT.FFT1D(FFTSource, true);
   int[] FFTResR = new int[FFTRes[0].length/avg];

   for(int i = 0;i<FFTRes[0].length/avg;i++){
      FFTResR[i] = 0;
      for(int j = 0;j<avg;j++){
      FFTResR[i] = FFTResR[i] + Math.abs((int)FFTRes[0][i*avg+j]);
      }
      FFTResR[i] = FFTResR[i] / avg;
      if (FFTResR[i] > Max) {Max = FFTResR[i];}
   }

   FFTMaxValue = Max;
   return FFTResR;
}

public static int[] f2m(int[] f){
   int[] m = new int[f2m(f.length)];
   for(int iCounter=0;iCounter<m.length;iCounter++)
   {m[iCounter]=f[m2f(iCounter)];}
   return m;
}

public static int[] logPower(int[] f){
```

```
   for(int iCounter=0;iCounter<f.length;iCounter++)
   {f[iCounter]=(int)Math.log(f[iCounter]);}
   return f;
}

public static int f2m(int f){
   return (int)(1127.01048*Math.log(1+f/700));
}

public static int m2f(int m){
   return (int)(700*(Math.exp(m/1127.01048)-1));
}

public static int SampleLength = 2048;
public static boolean isUseMelScale = false;
public static boolean isUseLogPower = false;
public static double SpectogramDeep = 0.5;
public static int FFTMaxValue;

}
```

9.2 Fourier transzformáció kódja

```
package segment;

import java.text.DecimalFormat;

/*
 VisAD system for interactive analysis and visualization of numerical
 data.  Copyright (C) 1996 - 2007 Bill Hibbard, Curtis Rueden, Tom
 Rink, Dave Glowacki, Steve Emmerson, Tom Whittaker, Don Murray, and
 Tommy Jasmin.

 This library is free software; you can redistribute it and/or
 modify it under the terms of the GNU Library General Public
 License as published by the Free Software Foundation; either
 version 2 of the License, or (at your option) any later version.

 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 Library General Public License for more details.

 You should have received a copy of the GNU Library General Public
```

```
 License along with this library; if not, write to the Free
 Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
 MA 02111-1307, USA
 */

/**
 * FFT is the VisAD class for Fourier Transforms, using the Fast Fourier
 * Transform when the domain length is a power of two.
 * <p>
 */

public class FFT {

       /**
        * compute 2-D Fourier transform, calling 1-D FT twice use FFT if rows and
        * cols are powers of 2
        *
        * @param rows
        *            first dimension for 2-D
        * @param cols
        *            second dimension for 2-D
        * @param x
        *            array for take Fourier transform of, dimensioned [2][length]
        *            where length = rows * cols, and the first index (2) is over
        *            real & imaginary parts
        * @param forward
        *            true for forward and false for backward
        * @return Fourier transform of x
        * @throws VisADException
        *             a VisAD error occurred
        */
       public static float[][] FT2D(int rows, int cols, float[][] x,
                     boolean forward) throws IllegalArgumentException {
              if (x == null)
                     return null;
              if (x.length != 2 || x[0].length != x[1].length) {
                     throw new IllegalArgumentException("bad x lengths");
              }
              int n = x[0].length;
              if (rows * cols != n) {
                     throw new IllegalArgumentException(rows + " * " + cols + " must
equal " + n);
              }
              float[][] y = new float[2][n];
```

```
        float[][] z = new float[2][rows];
        for (int c = 0; c < cols; c++) {
                int i = c * rows;
                for (int r = 0; r < rows; r++) {
                        z[0][r] = x[0][i + r];
                        z[1][r] = x[1][i + r];
                }
                z = FT1D(z, forward);
                for (int r = 0; r < rows; r++) {
                        y[0][i + r] = z[0][r];
                        y[1][i + r] = z[1][r];
                }
        }

        float[][] u = new float[2][n];
        float[][] v = new float[2][cols];
        for (int r = 0; r < rows; r++) {
                int i = r;
                for (int c = 0; c < cols; c++) {
                        v[0][c] = y[0][i + c * rows];
                        v[1][c] = y[1][i + c * rows];
                }
                v = FT1D(v, forward);
                for (int c = 0; c < cols; c++) {
                        u[0][i + c * rows] = v[0][c];
                        u[1][i + c * rows] = v[1][c];
                }
        }
        return u;
}

/**
 * compute 2-D Fourier transform, calling 1-D FT twice use FFT if rows and
 * cols are powers of 2
 *
 * @param rows
 *            first dimension for 2-D
 * @param cols
 *            second dimension for 2-D
 * @param x
 *            array for take Fourier transform of, dimensioned [2][length]
 *            where length = rows * cols, and the first index (2) is over
 *            real & imaginary parts
 * @param forward
```

```
         *            true for forward and false for backward
         * @return Fourier transform of x
         * @throws VisADException
         *             a VisAD error occurred
         */
        public static double[][] FT2D(int rows, int cols, double[][] x,
                    boolean forward) throws IllegalArgumentException {
              if (x == null)
                    return null;
              if (x.length != 2 || x[0].length != x[1].length) {
                    throw new IllegalArgumentException("bad x lengths");
              }
              int n = x[0].length;
              if (rows * cols != n) {
                    throw new IllegalArgumentException(rows + " * " + cols + " must
equal " + n);
              }
              double[][] y = new double[2][n];
              double[][] z = new double[2][rows];
              for (int c = 0; c < cols; c++) {
                    int i = c * rows;
                    for (int r = 0; r < rows; r++) {
                          z[0][r] = x[0][i + r];
                          z[1][r] = x[1][i + r];
                    }
                    z = FT1D(z, forward);
                    for (int r = 0; r < rows; r++) {
                          y[0][i + r] = z[0][r];
                          y[1][i + r] = z[1][r];
                    }
              }

              double[][] u = new double[2][n];
              double[][] v = new double[2][cols];
              for (int r = 0; r < rows; r++) {
                    int i = r;
                    for (int c = 0; c < cols; c++) {
                          v[0][c] = y[0][i + c * rows];
                          v[1][c] = y[1][i + c * rows];
                    }
                    v = FT1D(v, forward);
                    for (int c = 0; c < cols; c++) {
                          u[0][i + c * rows] = v[0][c];
                          u[1][i + c * rows] = v[1][c];
```

```
            }
        }
        return u;
}

/**
 * compute 1-D Fourier transform use FFT if length (2nd dimension of x) is a
 * power of 2
 *
 * @param x
 *            array for take Fourier transform of, dimensioned [2][length],
 *            the first index (2) is over real & imaginary parts
 * @param forward
 *            true for forward and false for backward
 * @return Fourier transform of x
 * @throws VisADException
 *             a VisAD error occurred
 */
public static float[][] FT1D(float[][] x, boolean forward) {
        if (x == null)
                return null;
        if (x.length != 2 || x[0].length != x[1].length) {
                throw new IllegalArgumentException("bad x lengths");
        }
        int n = x[0].length;
        int n2 = 1;
        boolean fft = true;
        while (n2 < n) {
                n2 *= 2;
                if (n2 > n) {
                        fft = false;
                }
        }
        if (fft)
                return FFT1D(x, forward);

        float[][] temp = new float[2][n];
        float angle = (float) (-2.0 * Math.PI / n);
        if (!forward)
                angle = -angle;
        for (int i = 0; i < n; i++) {
                temp[0][i] = (float) Math.cos(i * angle);
                temp[1][i] = (float) Math.sin(i * angle);
        }
```

```
	float[][] y = new float[2][n];
	for (int i = 0; i < n; i++) {
		float re = 0.0f;
		float im = 0.0f;
		for (int j = 0; j < n; j++) {
			int m = (i * j) % n;
			re += x[0][j] * temp[0][m] - x[1][j] * temp[1][m];
			im += x[0][j] * temp[1][m] + x[1][j] * temp[0][m];
		}
		y[0][i] = re;
		y[1][i] = im;
	}
	if (!forward) {
		for (int i = 0; i < n; i++) {
			y[0][i] /= n;
			y[1][i] /= n;
		}
	}
	return y;
}

/**
 * compute 1-D FFT transform length (2nd dimension of x) must be a power of
 * 2
 *
 * @param x
 *            array for take Fourier transform of, dimensioned [2][length],
 *            the first index (2) is over real & imaginary parts
 * @param forward
 *            true for forward and false for backward
 * @return Fourier transform of x
 * @throws VisADException
 *             a VisAD error occurred
 */
public static float[][] FFT1D(float[][] x, boolean forward) {
	if (x == null)
		return null;
	if (x.length != 2 || x[0].length != x[1].length) {
		throw new IllegalArgumentException("bad x lengths");
	}
	int n = x[0].length;
	int n2 = 1;
	while (n2 < n) {
		n2 *= 2;
```

```
            if (n2 > n) {
                  throw new IllegalArgumentException(
                              "x length must be power of 2");
            }
      }
      n2 = n / 2;
      float[][] temp = new float[2][n2];
      float angle = (float) (-2.0 * Math.PI / n);
      if (!forward)
            angle = -angle;
      for (int i = 0; i < n2; i++) {
            temp[0][i] = (float) Math.cos(i * angle);
            temp[1][i] = (float) Math.sin(i * angle);
      }
      float[][] y = FFT1D(x, temp);
      if (!forward) {
            for (int i = 0; i < n; i++) {
                  y[0][i] /= n;
                  y[1][i] /= n;
            }
      }
      return y;
}

/** inner function for 1-D Fast Fourier Transform */
private static float[][] FFT1D(float[][] x, float[][] temp) {
      int n = x[0].length;
      int n2 = n / 2;
      int k = 0;
      int butterfly;
      int buttered = 0;
      if (n == 1) {
            float[][] z1 = { { x[0][0] }, { x[1][0] } };
            return z1;
      }

      butterfly = (temp[0].length / n2);

      float[][] z = new float[2][n2];
      float[][] w = new float[2][n2];

      for (k = 0; k < n / 2; k++) {
            int k2 = 2 * k;
            z[0][k] = x[0][k2];
```

```
                z[1][k] = x[1][k2];
                w[0][k] = x[0][k2 + 1];
                w[1][k] = x[1][k2 + 1];
        }

        z = FFT1D(z, temp);
        w = FFT1D(w, temp);

        float[][] y = new float[2][n];
        for (k = 0; k < n2; k++) {
                y[0][k] = z[0][k];
                y[1][k] = z[1][k];

                float re = w[0][k] * temp[0][buttered] - w[1][k]
                                * temp[1][buttered];
                float im = w[0][k] * temp[1][buttered] + w[1][k]
                                * temp[0][buttered];
                w[0][k] = re;
                w[1][k] = im;

                y[0][k] += w[0][k];
                y[1][k] += w[1][k];
                y[0][k + n2] = z[0][k] - w[0][k];
                y[1][k + n2] = z[1][k] - w[1][k];
                buttered += butterfly;
        }
        return y;
}

/**
 * compute 1-D Fourier transform use FFT if length (2nd dimension of x) is a
 * power of 2
 *
 * @param x
 *            array for take Fourier transform of, dimensioned [2][length],
 *            the first index (2) is over real & imaginary parts
 * @param forward
 *            true for forward and false for backward
 * @return Fourier transform of x
 * @throws VisADException
 *             a VisAD error occurred
 */
public static double[][] FT1D(double[][] x, boolean forward)
                throws IllegalArgumentException {
```

```
if (x == null)
	return null;
if (x.length != 2 || x[0].length != x[1].length) {
	throw new IllegalArgumentException("bad x lengths");
}
int n = x[0].length;
int n2 = 1;
boolean fft = true;
while (n2 < n) {
	n2 *= 2;
	if (n2 > n) {
		fft = false;
	}
}
if (fft)
	return FFT1D(x, forward);

double[][] temp = new double[2][n];
double angle = -2.0 * Math.PI / n;
if (!forward)
	angle = -angle;
for (int i = 0; i < n; i++) {
	temp[0][i] = Math.cos(i * angle);
	temp[1][i] = Math.sin(i * angle);
}
double[][] y = new double[2][n];
for (int i = 0; i < n; i++) {
	double re = 0.0;
	double im = 0.0;
	for (int j = 0; j < n; j++) {
		int m = (i * j) % n;
		re += x[0][j] * temp[0][m] - x[1][j] * temp[1][m];
		im += x[0][j] * temp[1][m] + x[1][j] * temp[0][m];
	}
	y[0][i] = re;
	y[1][i] = im;
}
if (!forward) {
	for (int i = 0; i < n; i++) {
		y[0][i] /= n;
		y[1][i] /= n;
	}
}
return y;
```

```
}

/**
 * compute 1-D FFT transform length (2nd dimension of x) must be a power of
 * 2
 *
 * @param x
 *            array for take Fourier transform of, dimensioned [2][length],
 *            the first index (2) is over real & imaginary parts
 * @param forward
 *            true for forward and false for backward
 * @return Fourier transform of x
 * @throws VisADException
 *             a VisAD error occurred
 */
public static double[][] FFT1D(double[][] x, boolean forward)
              throws IllegalArgumentException {
       if (x == null)
              return null;
       if (x.length != 2 || x[0].length != x[1].length) {
              throw new IllegalArgumentException("bad x lengths");
       }
       int n = x[0].length;
       int n2 = 1;
       while (n2 < n) {
              n2 *= 2;
              if (n2 > n) {
                     throw new IllegalArgumentException(
                                   "x length must be power of 2");
              }
       }
       n2 = n / 2;
       double[][] temp = new double[2][n2];
       double angle = (double) (-2.0 * Math.PI / n);
       if (!forward)
              angle = -angle;
       for (int i = 0; i < n2; i++) {
              temp[0][i] = (double) Math.cos(i * angle);
              temp[1][i] = (double) Math.sin(i * angle);
       }
       double[][] y = FFT1D(x, temp);
       if (!forward) {
              for (int i = 0; i < n; i++) {
                     y[0][i] /= n;
```

```
                        y[1][i] /= n;
                }
        }
        return y;
}

/** inner function for 1-D Fast Fourier Transform */
private static double[][] FFT1D(double[][] x, double[][] temp) {
        int n = x[0].length;
        int n2 = n / 2;
        int k = 0;
        int butterfly;
        int buttered = 0;
        if (n == 1) {
                double[][] z1 = { { x[0][0] }, { x[1][0] } };
                return z1;
        }

        butterfly = (temp[0].length / n2);

        double[][] z = new double[2][n2];
        double[][] w = new double[2][n2];

        for (k = 0; k < n2; k++) {
                int k2 = 2 * k;
                z[0][k] = x[0][k2];
                z[1][k] = x[1][k2];
                w[0][k] = x[0][k2 + 1];
                w[1][k] = x[1][k2 + 1];
        }

        z = FFT1D(z, temp);
        w = FFT1D(w, temp);

        double[][] y = new double[2][n];
        for (k = 0; k < n2; k++) {
                y[0][k] = z[0][k];
                y[1][k] = z[1][k];

                double re = w[0][k] * temp[0][buttered] - w[1][k]
                                * temp[1][buttered];
                double im = w[0][k] * temp[1][buttered] + w[1][k]
                                * temp[0][buttered];
                w[0][k] = re;
```

```
                w[1][k] = im;

                y[0][k] += w[0][k];
                y[1][k] += w[1][k];
                y[0][k + n2] = z[0][k] - w[0][k];
                y[1][k + n2] = z[1][k] - w[1][k];
                buttered += butterfly;
            }
            return y;
        }

        /** test Fourier Transform methods */
        public static void main(String args[]) throws IllegalArgumentException {
            int n = 16;
            int rows = 1, cols = 1;
            boolean twod = false;

            DecimalFormat format = new DecimalFormat("0.0000");

            /*
             * if (args.length > 0 && args[0].startsWith("AREA")) { DisplayImpl
             * display1 = new DisplayImplJ3D("display"); DisplayImpl display1 =
new
             * DisplayImplJ3D("display"); AreaAdapter areaAdapter = new
             * AreaAdapter(args[0]); Data image = areaAdapter.getData();
             * FunctionType imageFunctionType = (FunctionType) image.getType();
             * RealType radianceType = (RealType) ((RealTupleType)
             * imageFunctionType.getRange()).getComponent(0); display.addMap(new
             * ScalarMap(RealType.Latitude, Display.Latitude)); display.addMap(new
             * ScalarMap(RealType.Longitude, Display.Longitude)); ScalarMap rgbMap
=
             * new ScalarMap(radianceType, Display.RGB); display.addMap(rgbMap); }
             */

            if (args.length > 0)
                n = Integer.valueOf(args[0]).intValue();
            if (args.length > 1) {
                rows = Integer.valueOf(args[0]).intValue();
                cols = Integer.valueOf(args[1]).intValue();
                n = rows * cols;
                twod = true;
            }

            float[][] x = new float[2][n];
```

```
            // double[][] x = new double[2][n];
            System.out.println("  initial values");
            if (twod) {
                  int i = 0;
                  for (int c = 0; c < cols; c++) {
                        for (int r = 0; r < rows; r++) {
                              x[0][i] = (float) (Math.sin(2 * Math.PI * r /
rows) * Math
                                          .sin(2 * Math.PI * c / cols));
                              x[1][i] = 0.0f;
                              // x[0][i] = (Math.sin(2 * Math.PI * r / rows) *
                              // Math.sin(2 * Math.PI * c / cols));
                              // x[1][i] = 0.0;
                              System.out.println("x[" + r + "][" + c + "] = "
                                          + format.format(x[0][i]) + " "
                                          + format.format(x[1][i]));
                              i++;
                        }
                  }
            } else {
                  for (int i = 0; i < n; i++) {
                        x[0][i] = (float) Math.sin(2 * Math.PI * i / n);
                        x[1][i] = 0.0f;
                        // x[0][i] = Math.sin(2 * Math.PI * i / n);
                        // x[1][i] = 0.0;
                        System.out.println("x[" + i + "] = " +
format.format(x[0][i])
                                    + " " + format.format(x[1][i]));
                  }
            }
            x = twod ? FT2D(rows, cols, x, true) : FT1D(x, true);
            System.out.println("\n  fft");
            if (twod) {
                  int i = 0;
                  for (int c = 0; c < cols; c++) {
                        for (int r = 0; r < rows; r++) {
                              System.out.println("x[" + r + "][" + c + "] = "
                                          + format.format(x[0][i]) + " "
                                          + format.format(x[1][i]));
                              i++;
                        }
                  }
            } else {
                  for (int i = 0; i < n; i++) {
```

```
                        System.out.println("x[" + i + "] = " +
format.format(x[0][i])
                                        + " " + format.format(x[1][i]));
                }
            }
            x = twod ? FT2D(rows, cols, x, false) : FT1D(x, false);
            System.out.println("\n  back fft");
            if (twod) {
                int i = 0;
                for (int c = 0; c < cols; c++) {
                    for (int r = 0; r < rows; r++) {
                        System.out.println("x[" + r + "][" + c + "] = "
                                + format.format(x[0][i]) + " "
                                + format.format(x[1][i]));
                        i++;
                    }
                }
            } else {
                for (int i = 0; i < n; i++) {
                    System.out.println("x[" + i + "] = " +
format.format(x[0][i])
                                        + " " + format.format(x[1][i]));
                }
            }
        }
}
```

9.3 A diszkrét koszinusz transzformáció kódja

```
/*File ForwardDCT01.java
Copyright 2006, R.G.Baldwin
Rev 01/03/06

The static method named transform performs a forward
Discreet Cosine Transform (DCT) on an incoming time series
and returns the DCT spectrum.

See http://en.wikipedia.org/wiki/Discrete_cosine_transform
#DCT-II and http://rkb.home.cern.ch/rkb/AN16pp/node61.html
for background on the DCT.

This formulation is from
http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/
coding/transform/dct.html
```

```
Incoming parameters are:
  double[] x - incoming real data
  double[] y - outgoing real data

Tested using J2SE 5.0 under WinXP.  Requires J2SE 5.0 or
later due to the use of static import of Math class.
**********************************************************/
import static java.lang.Math.*;

public class ForwardDCT01{

  public static void transform(double[] x,
                               double[] y){

    int N = x.length;

    //Outer loop interates on frequency values.
    for(int k=0; k < N;k++){
      double sum = 0.0;
      //Inner loop iterates on time-series points.
      for(int n=0; n < N; n++){
        double arg = PI*k*(2.0*n+1)/(2*N);
        double cosine = cos(arg);
        double product = x[n]*cosine;
        sum += product;
      }//end inner loop

      double alpha;
      if(k == 0){
        alpha = 1.0/sqrt(2);
      }else{
        alpha = 1;
      }//end else
      y[k] = sum*alpha*sqrt(2.0/N);
    }//end outer loop
  }//end transform method
  //--------------------------------------------------//
}//end class ForwardDCT01
```

10 Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani Dr. Hunyadi László tanár úrnak a dolgozatom elkészítéséhez nyújtott segítségéért, ötleteiért. Szeretnék még köszönetet mondani családomnak támogatásukért. Különösen három hónapos kislányomnak, aki éjszakai alvásaival nagymértékben hozzájárult a dolgozat befejezéséhez. Valamint szeretnék köszönetet mondani mindenkinek, akik segítettek dolgozatom megvalósításában.